Package: cpp11 (via r-universe)

July 26, 2024

Title A C++11 Interface for R's C Interface

Version 0.4.7.9000

Description Provides a header only, C++11 interface to R's C interface. Compared to other approaches 'cpp11' strives to be safe against long jumps from the C API as well as C++ exceptions, conform to normal R function semantics and supports interaction with 'ALTREP' vectors.

License MIT + file LICENSE

URL https://cpp11.r-lib.org, https://github.com/r-lib/cpp11

BugReports https://github.com/r-lib/cpp11/issues

Depends R (>= 3.5.0)

Suggests bench, brio, callr, cli, covr, decor, desc, ggplot2, glue, knitr, lobstr, mockery, progress, rmarkdown, scales, Rcpp, testthat (>= 3.2.0), tibble, utils, vctrs, withr

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Config/Needs/cpp11/cpp_register brio, cli, decor, desc, glue, tibble, vctrs

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Repository https://fastverse.r-universe.dev

RemoteUrl https://github.com/r-lib/cpp11

RemoteRef HEAD

RemoteSha aeb67b3f354152974cd06d5c6af7145e6ddd0717

Contents

`cpp_register` *Generates wrappers for registered C++ functions*

Description

Functions decorated with `[[cpp11::register]]` in files ending in `.cc`, `.cpp`, `.h` or `.hpp` will be wrapped in generated code and registered to be called from R.

Usage

```
cpp_register(
  path = ".",
  quiet = !is_interactive(),
  extension = c(".cpp", ".cc")
)
```

Arguments

`path`	The path to the package root directory
`quiet`	If `TRUE` suppresses output from this function
`extension`	The file extension to use for the generated src/cpp11 file. `.cpp` by default, but `.cc` is also supported.

Details

Note registered functions will not be *exported* from your package unless you also add a `@export` roxygen2 directive for them.

In order to use `cpp_register()` the `cli`, `decor`, `desc`, `glue`, `tibble` and `vctrs` packages must also be installed.

Value

The paths to the generated R and C++ source files (in that order).

Examples

```
# create a minimal package
dir <- tempfile()
dir.create(dir)

writeLines("Package: testPkg", file.path(dir, "DESCRIPTION"))
writeLines("useDynLib(testPkg, .registration = TRUE)", file.path(dir, "NAMESPACE"))

# create a C++ file with a decorated function
dir.create(file.path(dir, "src"))
writeLines("[[cpp11::register]] int one() { return 1; }", file.path(dir, "src", "one.cpp"))

# register the functions in the package
cpp_register(dir)

# Files generated by registration
file.exists(file.path(dir, "R", "cpp11.R"))
file.exists(file.path(dir, "src", "cpp11.cpp"))

# cleanup
unlink(dir, recursive = TRUE)
```

---

cpp_source *Compile C++ code*

Description

cpp_source() compiles and loads a single C++ file for use in R. cpp_function() compiles and loads a single function for use in R. cpp_eval() evaluates a single C++ expression and returns the result.

Usage

```
cpp_source(
  file,
  code = NULL,
  env = parent.frame(),
  clean = TRUE,
  quiet = TRUE,
  cxx_std = Sys.getenv("CXX_STD", "CXX11"),
  dir = tempfile()
)

cpp_function(
  code,
  env = parent.frame(),
  clean = TRUE,
  quiet = TRUE,
```

```
  cxx_std = Sys.getenv("CXX_STD", "CXX11")
)

cpp_eval(
  code,
  env = parent.frame(),
  clean = TRUE,
  quiet = TRUE,
  cxx_std = Sys.getenv("CXX_STD", "CXX11")
)
```

Arguments

file	A file containing C++ code to compile
code	If non-null, the C++ code to compile
env	The R environment where the R wrapping functions should be defined.
clean	If TRUE, cleanup the files after sourcing
quiet	If 'TRUE', do not show compiler output
cxx_std	The C++ standard to use, the CXX_STD make macro is set to this value. The default value queries the CXX_STD environment variable, or uses 'CXX11' if unset.
dir	The directory to store the generated source files. tempfile() is used by default. The directory will be removed if clean is TRUE.

Details

Within C++ code you can use [[cpp11::linking_to("pkgxyz")]] to link to external packages. This is equivalent to putting those packages in the LinkingTo field in a package DESCRIPTION.

Value

For cpp_source() and [cpp_function()] the results of dyn.load() (invisibly). For [cpp_eval()] the results of the evaluated expression.

Examples

```
cpp_source(
  code = '#include "cpp11/integers.hpp"

  [[cpp11::register]]
  int num_odd(cpp11::integers x) {
    int total = 0;
    for (int val : x) {
      if ((val % 2) == 1) {
        ++total;
      }
    }
    return total;
  }
```

```
  ')

num_odd(as.integer(c(1:10, 15, 23)))

if (interactive() && require("progress")) {

cpp_source(
  code = '
#include <cpp11/R.hpp>
#include <RProgress.h>

[[cpp11::linking_to("progress")]]

[[cpp11::register]] void
show_progress() {
  RProgress::RProgress pb("Processing [:bar] ETA: :eta");

  pb.tick(0);
  for (int i = 0; i < 100; i++) {
    usleep(2.0 / 100 * 1000000);
    pb.tick();
  }
}
')

show_progress()
}
```

---

cpp_vendor *Vendor the cpp11 dependency*

Description

Vendoring is the act of making your own copy of the 3rd party packages your project is using. It is often used in the go language community.

Usage

```
cpp_vendor(path = ".")
```

Arguments

path	The path to the package root directory

Details

This function vendors cpp11 into your package by copying the cpp11 headers into the `inst/include` folder of your package and adding 'cpp11 version: XYZ' to the top of the files, where XYZ is the version of cpp11 currently installed on your machine.

If you choose to vendor the headers you should *remove* `LinkingTo: cpp11` from your DESCRIPTION.

Note: vendoring places the responsibility of updating the code on **you**. Bugfixes and new features in cpp11 will not be available for your code until you run `cpp_vendor()` again.

Value

The file path to the vendored code (invisibly).

Examples

```
# create a new directory
dir <- tempfile()
dir.create(dir)

# vendor the cpp11 headers into the directory
cpp_vendor(dir)

list.files(file.path(dir, "inst", "include", "cpp11"))

# cleanup
unlink(dir, recursive = TRUE)
```

Index